SICE 第15回システム工学部会研究
1994/07/18

発散的思考支援システム「Keyword Associator」第二版

渡部 勇
(株) 富士通研究所 情報社会科学研究所
isamu@iias.flab.fujitsu.co.jp

1 はじめに

発想の行きづまりを打開し，アイデアを広げていくためにはどうしたよいだろうか．一つの有効な手段は「外」からの刺激を受けることである．一人だけで考えるのをやめて，とりあえず誰かに相談してみる．すると，その人が出したアイデアがきっかけとなって，新たなアイデアが生まれる．グループで集まって考えてみる．すると，互いに刺激し合うことで相乗効果を生み，次々とアイデアが出てくる．むろん，いつでもこんなにうまくいくわけではないが，誰でも一度は経験があるのではないだろうか．きっかけとなるのは，必ずしも今考えている課題に対する「アイデア」でなくてもよい．本を読む．論文を読む．講演会を聞きにいく．そもそもの目的は，情報を得たり知識を増やしたりすることなのだが，読んだり聞いたりしているうちに，いろいろと考えをめぐらせ，思わぬアイデアに出合うこともある．要は，膠着してしまった状態に，刺激を加えることによって，揺さぶりをかけてやることが重要である．

Keyword Associator とは，アイデアを広げていくきっかけとなるような刺激を，計算機から得るためのツールである．Keyword Associator は，電子ニュース記事などのテキスト情報から，連想辞書を自動的に構築する．そして，ユーザが入力したキーワードやテキストに対し，連想辞書の中から関連するキーワードや関連するテキストを探し出し提示してくれる．システムから得られる情報がすべて役に立つわけではないが，中には自分だけでは到底思いつかないようなものも含まれている．これらが刺激となり，アイデアを広げていくこと，すなわち発散的思考を行なう際の助けになるわけである．

我々は，この「連想検索による発散的思考支援」というアイデア [1][2] の実現可能性を確認するために，最初のプロトタイプシステムである Keyword Associator 第〇版[1] を開発した [3]．その後，GUI(Graphical User Interface) を持つ第一版[2] を開発し [4]，現在，その試用経験をもとに第二版を開発している．本稿では，第二版で実現・強化された諸機能を中心に Keyword Associator の紹介を行なう．以下，第2章でシステム構成，第3章でユーザインタフェース，第4章で連想辞書構築，第5章で連想検索についてそれぞれ説明し，第二版と特徴ともいえる，以下の各改良点について述べていく．なお，括弧内の数字は説明がある節の番号を表している．

- サーバ・クライアント・アーキテクチャ (2.2)
- 他ツールとの連係 (2.3)
- 操作性の向上 (3.1, 3.2)
- 自動リンク付け (3.3)
- テキスト単位での統計処理 (4.3)
- 連想検索の高速化・省メモリ化 (4.3)
- 出現頻度のカウントの高精度化 (4.4)
- 連想辞書の日本語化 (4.5)
- 推移的連想 (5.2)
- 複数入力に対する重みの自動設定 (5.3)
- 複数入力に対する検索結果の結合 (5.4)
- ユーザへの適応 (5.5)
- 分野による検索結果のフィルタリング (5.6)
- 品詞による検索結果のフィルタリング (5.7)

[1] 第〇版を作成した時点では，まだ Keyword Associator という名前はつけられていなかった．第〇版のユーザインタフェースはテキストベースのものであった．

[2] X ウィンドウシステム用のアプリケーションとして，XView2 ツールキットを使用して作成された．

2 システム構成

図 1に Keyword Associator 第二版のシステム構成を示す．図に示すように，Keyword Associator は，

- kaeditor
- katool
- kaserver
- kabuilder

という 4 つのサブシステムから構成されており，それぞれが独立したプログラムとして実現されている．

本章では，まず，この 4 つのサブシステムの役割とシステム全体の処理の流れを説明したのち，第二版において上記のようなシステム構成を採用した理由を述べる[3]．なお，各サブシステムの仕組み・機能・使い方に関しては，kaeditor と katool は第 3章，kabuilder は第 4章，kaserver は第 5章において，それぞれ詳細な説明を行なう．

2.1 4 つのサブシステム

kaeditor は，発想を行なうための作業エリアを提供するエディタであり，「黒板」あるいは「メモ帳」の役割を果たす．キーワードや画像を入力し，二次元平面上で自由に配置したり，その間をリンクで結んだりすることができる．

katool は，検索のためのユーザインタフェースを提供するクライアントプログラムであり，検索入力，検索結果の表示，検索条件の設定，作業履歴などの個人情報の管理に利用される．

kaeditor と katool は，X ウィンドウシステムのクライアント間通信メカニズムにより双方向通信を行なう．kaeditor 上のキーワードを katool の入力としたり，katool 上に表示される検索結果を kaeditor にコピーして張り付けたりすることができる．

kaserver は，連想辞書を用いて連想検索を行なうサーバプログラムである．kaserver と katool は，UNIX のプロセス間通信メカニズムにより双方向通信を行なっており，検索入力は katool から kaserver へと渡される．kaserver では以下の 4 通りの検索が可能であり，検索結果は，いずれの場合も関連の近い順に並べられ，katool へと転送され表示される．

検索方法 1: キーワードを入力し，関連キーワードを得る
検索方法 2: キーワードを入力し，関連テキストを得る
検索方法 3: テキストを入力し，関連キーワードを得る
検索方法 4: テキストを入力し，関連テキストを得る

kabuilder は，テキスト情報から連想辞書を構築するためのプログラムである．情報ソースとして利用するテキスト情報は，計算機可読であればなんでもよい．今までに実際に試してみたものとしては，電子ニュースの記事，オンラインマニュアル，特許明細書，百人一首などがあり，情報ソースを変えることにより，一般的な連想辞書から専門的なものまで，さまざまなタイプの連想辞書を構築することができる．

[3] 第一版においては，kaeditor, katool, kaserver に相当する部分が単一のプログラムとして実現されていた．

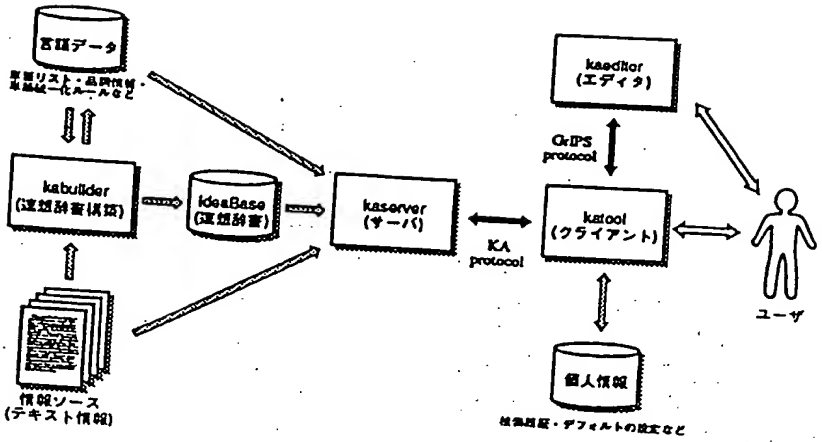

図 1: Keyword Associator のシステム構成
(図中，黒い矢印はファイルを介したデータの流れを，グレーの矢印は通信によるデータの流れを表す)

中でも，電子ニュース記事は，発散的思考支援に適した連想辞書を構築するための情報ソースとして，以下のような特徴を持っている [4].

- 大規模
- 広い分野をカバー
- 最新の話題を反映
- 通常の辞書にはない自由な連想関係

2.2 サーバ・クライアント・アーキテクチャ

近年の Internet の発展にともない，電子ニュースの流通量 (投稿量) は年々増大してきている．さらに，CD-ROM の普及などもあり，大量でかつ多様なテキスト情報を，計算機可読の形で容易に手に入れることができるようになってきている．Keyword Associator は，テキスト情報から大規模な連想辞書を構築し，それを利用することを特徴としており，こういった傾向はむろん望ましいことではある．が，一方で，連想辞書が巨大化[4] することは，必要とする計算機資源 (ディスク・メモリ・実行時間など) の増加，パフォーマンスの低下につながる．Keyword Associator 第二版では，データ構造・アルゴリズムの改善などを行なうとともに，kaserver と katool からなるサーバ・クライアント・アーキテクチャを採用することにより，このような問題の解決をはかっている．

複数人での同時使用による計算機資源の有効利用

kaserver は，複数の katool と通信することができる．また，kaserver と katool は，ネットワークでつながっていれば，別々の計算機上で実行することもできる．したがって，メモリを多く積んだ CPU の速い計算機上で kaserver を実行し，各ユーザが使用している計算機上で動いている複数の katool からの検索要求を処理させることができる．このように，一つの kaserver を複数人で共有することによって，計算機資源を有効に利用することが可能になる．

複数連想辞書の同時使用による負荷分散

katool は，複数の kaserver と通信することができる．大規模な連想辞書の場合，それを分割するか[5]，あるいは，5.6 節で説明する分野によるフィルタリングを施すことにより，複数の小規模な連想辞書として扱うことができる．この複数の連想辞書を別々の計算機上で動いている別々の kaserver によって検索させることにより[6]，計算機の負荷を分散し，パフォーマンスを向上させることが可能になる．このような構成での利用は，大規模な連想辞書を用いる場合だけでなく，多種多様な連想辞書を同時に用いるときにも有効である．

ユーザから見た立ち上げ時間の短縮

kaserver と katool を，一対一で，かつ，同じ計算機上で実行する場合でも，ユーザから見たシステムの立ち上げ時間が短縮するという利点が得られる．Keyword Associator では，パフォーマンスを向上するために，連想辞書の一部または全部をメモリ上に読み込む．特に第一版では，立ち上げ時に連想辞書をすべてメモリ上に展開するため，起動してから実際に使えるようになるまでに結構時間がかかり，使いたいときにすぐに使えないという不満があった．第二版でも，kaserver に関しては，第一版に比べれば大幅に改善しているものの立ち上げにはやはり若干の時間を要する．しかし，katool は一瞬で立ち上がるため，kaserver を常に動かしている状態にしておけば，ユーザが使いたいと思ったときにすぐ，システムを利用することが可能になる．

2.3 他ツールとの連係

Keyword Associator 第二版では，連想検索を行なうための katool と kaserver を，発想作業を行なうためのエディタである kaeditor と切りわけ，別々のプログラムとして実現することにより，Keyword Associator 以外のさまざまなツールと組み合わせて利用することができるようになっている．

グループ発想支援システム GrIPS

Keyword Associator は，グループ発想支援システム GrIPS[5] のサブシステムとして，発散的思考を支援する役割を担っている．ただし，発想作業を行なうためのエディタとして，kaeditor の代わりに，kaeditor をベースにグループで使用できるように発展させた共有黒板ツール SharedBoard を使用する．GrIPS の各サブシステムの間では，kaeditor と katool の間の通信方式を拡張した GrIPS プロトコル (図 1) により，データの受渡しを行なう．この通信機能により，katool を，SharedBoard はもちろんのこと，他の GrIPS サブシステムと組み合わせて利用することができる．

[4] 第〇版では，約 70MB のテキスト情報から構築した約 1 万語の連想辞書，第一版では，約 200MB のテキスト情報から構築した約 6 万語の連想辞書を使用していた．第二版では，最大のものでは，約 1.5 GB のテキスト情報から構築した約 15 万語の連想辞書を使用している．

[5] 連想辞書を分割・マージするためのユーティリティプログラムがある．

[6] 第二版では，一つの kaserver で複数の連想辞書を扱うこともできる．

Xアプリケーションとの連係

Keyword Associatorでは，Xウィンドウシステムで標準になっているコピー&ペーストによるデータの受渡しもサポートしている．したがって，katoolを，例えばemacsなどのテキストエディタとともに利用することも可能である．

3 ユーザインタフェース

Keyword Associatorの4つのサブシステムのうち，ユーザが直接利用するのはkaeditorとkatoolの2つ[7]である．本章では，この2つのサブシステムのユーザインタフェースの話題を中心に，Keyword Associatorを使ってどのように発想を進めていくのかを，実行例を交えながら説明していく．

3.1 kaeditor

図2の左側にあるのがkaeditorである．これが発想作業を進めていくための作業エリアになる．まずは，ここに思いついたキーワードを次々と入力していく．キーワードは自由な位置に配置することができる．また，キーワードの間にリンクを張ることもできる．連想のパスにしたがってリンクを張っておけば，あとで見たときに，発想がどのように広がっていったのかが一目でわかるようになる．

アイデアは，文章として入力するのではなく，アイデアの断片を表すキーワードを入力することによって表現する．発散的思考の段階においては，アイデアあるいはアイデアの断片を組み合わせることによって，新たなアイデアが生まれることがよくある．文章ではなくキーワードとして断片の形で入力するのは，この組合せを容易にするためである．

3.2 katool

発想に行きづまったら，図2の右側にあるkatoolを用いて連想検索を行なう．図は，「知的 データベース」というキーワードを入力し，それに対する検索結果が表示されたところを表している．

検索のための入力は，以下の3通りの方法により行なうことができる．

ドラッグ&ドロップ
: kaeditor上のキーワードをマウスでドラッグし，katool上にドロップする(マウスボタンを離す)．ドラッグ&ドロップによる入力は，GrIPSの他のサブシステムやOPEN LOOK仕様のアプリケーションからでも行なうことができる．また，katoolの中に表示されている検索結果をドラッグ&ドロップし，入力とすることも可能である．

コピー&ペースト
: emacsなどのXアプリケーションと組み合わせて使用する場合には，アプリケーション上でマウスにより文字列を選択し，katool上にカーソルを移動してからPasteキーを押す．すると選択された文字列がコピー&ペーストされ，入力となる．

キーボード入力
: katoolの上段にある横長のウィンドウが，入力エリアになっている．ここにカーソルを持っていけば，キーボードから入力することもできる．

2.1節で述べたように，Keyword Associatorには，キーワードとテキストを組み合わせた4通りの検索方法が用意されている．このうちどの検索を行なうかは，以下に示すように，入力を行なった場所[8]によって判断される．なお，テキストの入力は，関連テキストエリアに表示された検索結果をドラッグ&ドロップすることによってのみ，行なうことができる．

入力	入力場所	検索方法
キーワード	入力エリア	方法1と2
キーワード	関連キーワードエリア	方法1
キーワード	関連テキストエリア	方法2
テキスト	入力エリア	方法3と4
テキスト	関連キーワードエリア	方法3
テキスト	関連テキストエリア	方法4

検索結果は，いずれの検索方法の場合でも，関連の強いものから順に並べられ，関連キーワードは中段左側にある関連キーワードエリアに，関連テキストは下段にある関連テキストエリアに，それぞれリスト形式で表示される．リストにおいて，左側の番号は順位，真中のゲージは関連の強さ(関連度)を表している．

なお，関連テキストエリアに表示されているのは，各テキストのタイトルであり，内容を見たいときには，タイトル上でマウスボタンをクリックする．すると，新たにウィンドウが開いてテキストの中身を読むことができる．なお，電子ニュースの場合には，記事のサブジェクトがタイトルになっている．

katoolの中段右側にあるのは，kaserverの設定を変更し，検索条件を変えるためのコントロールパネルである．katoolの実行中にしばしば変更する可能性のある設定項目が，コントロールパネル内に置かれている．その他，あまり頻繁に変えないものに関しては，メニューを選んで設定用のウィンドウを開き，その中で変更するようになっている．

3.3 連想検索による発散的思考支援

Keyword Associatorが提供する連想検索のうち，アイデアを広げるために利用するのは，主に検索方法1のキーワードとキーワードの間の連想関係である．あとの3つは，どちらかというと発散的思考の前に行なう情報収集を支援するためのものであり，情報検索的な使い方になる．これらの検索方法でも発散的思考の刺激を得ることは可能であるが，方法1に比べるとその効果は弱い．検索結果として関連テキストが得られる場合，なぜそのテキストが検索されたのかは，内容を見ればすぐにわかってしまう．一方，検索結果として関連キーワードが得られる場合には，なぜそれが選ばれてきたのかがよくわからないケースが多々ある．そして，そのようなときに，なぜ選ばれてきたのかを考えること，すなわち，自分が入力したキーワードと，システムが提示してきたキーワードの間の連想関係の背後に何があるのかを想像することが，アイデアを広げる大きなきっかけとなるのである．

システムが提示してくる関連キーワードには，ユーザの立場からみて以下のの4種類が考えられる．

case1: 思いもかけなかったもの[9]

case2: そのときには気がついていなかったもの[10]

case3: はじめてみたもの[11]

case4: 提示されることが十分に予想されたもの[12]

[7] kaeditorとkatoolは，ともにXView3ツールキットを使用し，C言語で書かれている．UNIX + Xウィンドウシステムという環境で動作する．

[8] ドラッグ&ドロップで入力した場合にはドロップした場所，コピー&ペーストで入力した場合にはPasteキーを押したときにマウスカーソルがあった場所がそれぞれ「入力場所」になる．キーボードを用いる場合には，入力エリアの中にしか入力することができない．

[9] 自分一人では考え付かないようなもの

[10] いずれは考え付いたであろうもの

[11] 意味のわからないもの

[12] 当たり前のもの

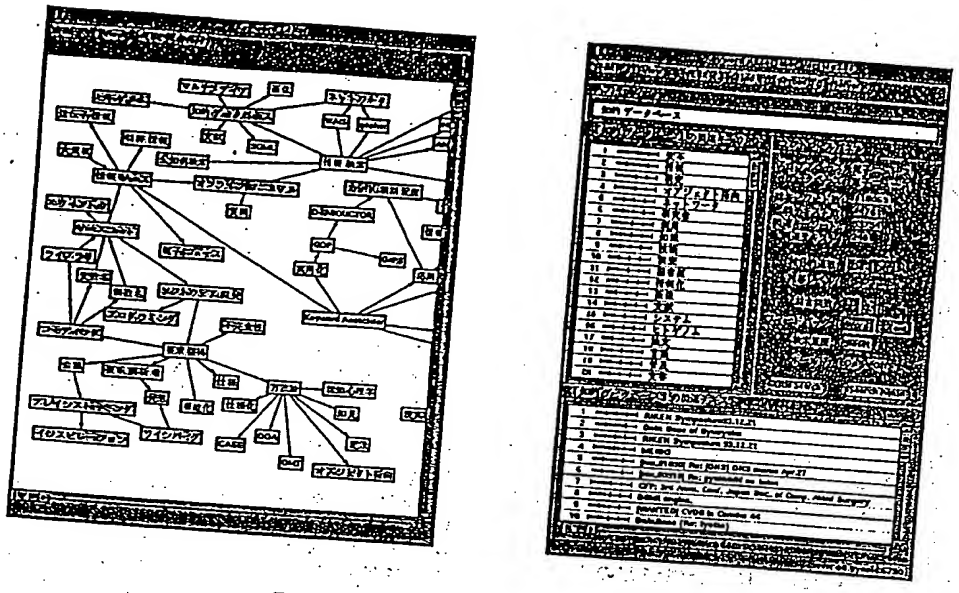

図 2: Keyword Associator の実行例
(約1カ月分の日本語の電子ニュース記事から構築した連想辞書を使用している)

case1 は，基本的にアイデアを広げるきっかけとなり得る．また，case2 は，刺激を受けるという意味では効果が弱いが，短時間のうちに網羅的にアイデアを出し尽くすという意味では非常に役に立つ．もし，提示されたキーワードが今考えている課題に対して直接役に立つものであれば，ドラッグ&ドロップによって kaeditor にコピーし，アイデアの断片に加える．この際，コピーされたキーワードと，もともと検索のために入力したキーワードの間には，自動的にリンクが張られ，発想のパスが記録として残される．提示されたキーワードが直接アイデアの断片にならなくても，そのキーワードがきっかけとなって新たなキーワードを思いつくこともある．思いついたキーワードは，kaeditor にキーボードから入力し，これもアイデアの断片に加える．提示されたキーワードを入力として与えることにより，次々と連想検索を進めていくことも可能である．このようにして，検索結果を利用しながら，簡単なマウス操作によってアイデアを広げていくことができる．

case3 は，新たな情報を得て知識を増やすという意味で役に立つ．意味がわからないキーワードが提示された場合には，そのキーワードに対する関連テキストを検索し，表示する．多くの場合，前後の文脈から大体の意味を推測することが可能であり，アイデアを広げていくための基礎知識を増やしていくことができる．

問題は，case4 である．ユーザにとって当たり前のものばかりが出てきてしまうと，それだけ役に立つものが減り，刺激を受けるチャンスが減ってしまう．当たり前のものはできるだけ提示されないようにしなければならない．この問題に関しては，5.5節のユーザへの適応においてまたあらためて説明することにする．

4 連想辞書構築

連想検索に使用される連想辞書は，Keyword Associator のサブシステムである kabuilder が，テキスト情報から統計処理により自動構築する．第一版までは，例えば電子ニュース記事のような大量のテキスト情報を入力とする場合には，各テキストをグループ化した上で統計処理を行なっていた．第二版では，これを各テキスト単位で行なうようにしている．

本章では，まず，連想辞書構築の手順について簡単に説明したあと，第一版において，なぜグループ単位での処理を行なっていたのか，第二版において，なぜテキスト単位で処理するようにしたのか，そして，そのためにどのような改良が必要であったのかを述べる．また，第二版において実現された，日本語のテキスト情報からの連想辞書構築の方法についても説明する．

4.1 連想辞書構築の手順

連想辞書は以下の手順で構築される．まず，各テキスト(グループ化した場合にはテキストグループ)に現れるすべての単語について，その出現頻度をカウントする．続いて，この出現頻度から，単語 w の，テキスト t における出現確率 $p_t(w)$ と，入力テキスト全体における出現確率 $p_g(w)$ をそれぞれ計算する．さらに，この $p_t(w)$ と $p_g(w)$ を用いて，テキスト t における単語 w の重要度 (Significance)[13]を，以下の式により計算する．

$$s_t(w) = p_t(w) \log \frac{p_t(w)}{p_g(w)}$$

この重要度 $s_t(w)$ を，すべてのテキスト，すべての単語について計算したものが，kaserver が連想検索の際に使用する連想辞書 IdeaBase となる．

4.2 グループ単位での統計処理

電子ニュースでは，階層構造を持ったニュースグループによってあらかじめ記事が分類されている．第一版までは，このニュースグループを統計処理の単位としており，

[13] 各単語について重要度の総和をとった，

$$\sum p_t(w) \log \frac{p_t(w)}{p_g(w)}$$

は，$p_g(w)$ の $p_t(w)$ からのダイバージェンス (Divergence，逸脱量) であり，テキスト全体での単語 w の使われ方のパターンとテキスト t でのパターンがどの程度異なっているのかを表す．重要度は，その違いにそれぞれの単語がどの程度寄与しているかを表すものであり，値の大きい単語は，そのテキストを特徴付ける (平均的なテキストから逸脱させる) 単語，すなわちキーワードであると考えることができる．

各ニュースグループ内の記事をすべて合わせたものを一つのテキストとして扱っていた．これには，以下のような理由があった．

まず第一の理由は，計算精度の問題である．一つ一つの電子ニュースの記事はあまり長くない．英語の電子ニュース記事の場合，中には非常に長い記事もあるが，数十ワードを中心にして，せいぜい数百ワードまでとなっている．記事単位で統計処理を行なった場合には，この記事のワード数が，重要度の計算で使用する $p_i(w)$ の分母になるため，計算精度が落ちてしまう．

第二の理由は，計算量に関するものである．連想検索においては，重要度のマトリックスを用いた計算を行なうため，テキストの数が増えれば，それにともなって，連想辞書の大きさ，検索に要する時間が増えていく．約1カ月分の英語の電子ニュース記事の場合，各ニュースグループには平均して数百個の記事が存在する．したがって，記事単位で処理をするようにした場合には，計算時間が大幅に増えるのはもちろんのこと，連想辞書をメモリ上に展開することができなくなり，さらにパフォーマンスが落ちてしまう．

第三の理由は，テキストのグループ化によってはじめて抽出される連想関係の存在である．同じニュースグループに属している記事の間には，似たような話題を扱っているという意味での関連性がある．したがって，別々の記事の中に現れる単語同士であっても，同一のニュースグループ内に存在している場合には，その間に何らかの連想関係があると考えることができる．Keyword Associator では，基本的には同一テキスト内に現れる単語間に連想関係があるものとして計算を行なうため，このような連想関係は，グループ単位で処理を行なわないと抽出されない．

4.3 テキスト単位での統計処理

ところが，第一版を試用していく中で，グループ単位での処理を行なっていることに起因する以下のような問題点が明らかになってきた．

第一の問題点は，連想関係の多様性に関するものである．a という単語と b という単語が，ともにテキストグループ i の中にしか現れないものとする．この場合，Keyword Associator における連想計算では，a に対する連想キーワードと b に対する連想キーワードとが，全く同じものになってしまう．ニュースグループには，そもそも似たような記事が集められているため，上記のような単語の出現分布の局所性が現れる．そのため，違うキーワードを入力したのに，よく似た結果や，あるいはまったく同じ結果が得られるような事態が起きてしまうことがあった．

第二の問題は，関連テキスト検索に関するものである．グループ単位での処理を行なった場合，関連テキスト検索の結果得られるのは，個々のテキストではなく，テキストのグループである．例えば，電子ニュースの場合は，ニュース記事ではなく，ニュースグループが得られることになる．第一版では，関連テキストグループ内の個々のテキストをテキストサーチプログラムにより検索し，テキストそのものを検索結果として得ることができるようになっていた．しかし，同一テキストグループ内のテキストは，入力キーワードに対しすべて同一の関連度を持つものとして提示されるため，3.3節の case3 のように，意味のわからないようなキーワード，したがって，あまり多く出現しないキーワードに関する関連テキストを検索する場合には役に立つものの，比較的出現回数が多いキーワードに対する検索を行なう場合には，あまり有効ではなかった．

第二版では，上記の2点を克服し，連想関係の多様性を増すことによって発散的思考支援システムとしての質を高めるとともに，情報検索・ブラウジング・フィルタリングなどの目的にも利用できるようにするために，大量のテキスト情報に対しても，各テキスト単位で統計処理を行なうようにした．その際，4.2節で述べたテキスト単位で処理を行なう場合の3つの問題点に対しては以下のように対処した．

まず第一の計算精度の問題であるが，予備実験を行なってみたところ，一つ一つのテキストの長さが短くなることによる精度の低下は，テキストの数の増加によって緩和される傾向にあることがわかった．ただし，単語の出現頻度のカウントの方法が精度に大きく影響するため，次節で述べるように kabuilder の改良を行なった．

第二の計算量の問題に関しては，kaserver のデータ構造・アルゴリズムを改善し，高速化・省メモリ化を行なった．また，連想計算で使用する転置マトリックスをあらかじめ計算して用意しておくことにより，検索実行時のパフォーマンス向上をはかっている．

第三のグループ化により抽出される連想関係の問題に関しては，同じサブジェクトのニュース記事をグループ化し，ニュースグループ単位で処理したものと組み合わせて使用することによって解決した．

4.4 出現頻度のカウント

電子ニュースの記事の中には，ニュースシステムが使用するための情報が書かれたヘッダ，投稿者が所属や名前などを表すために入れる signature，他の記事を引用する時に使用される記法[14]など，本文とは関係のないものが含まれており，単純な単語のカウントを行なうと，意味のないものが多くカウントされ連想検索に悪影響を及ぼすことになる．したがって，記事のうち本文以外の部分[15]をできるだけ飛ばして単語のカウントを行なうようにしている[16]．

ニュース記事では，他のテキスト情報に比べて，ミススペルの単語の割合がだいぶ多い．また，表記の揺れ，活用などにより，同一の単語が違う形で現れることも多い．単純な単語のカウントを行なうと，本来同じ単語であるものが別々にカウントされてしまい，連想関係が正しく計算されなくなる．そこで，以下に示すような単語の統一化を行なうことにより，できるだけ多くの連想関係が抽出されるようにしている[17]．ただし，単語の統一化もあまりやり過ぎると，かえって逆効果になるケースもあるため，まず，すべての単語を別々のものとしてカウントし，全テキストを読み終った段階で，各単語の頻度情報を参考にしながら単一化を行なう．

変化形: [18] keywords → keyword
派生語: [19] internationalizable → internationalization
複合語: guitarplayer → guitar player
米語／英語: colour → color
ミススペル: unfammilier → unfamiliar
接頭語: nonreflective → non reflective
大文字／小文字: Sparcstation → SPARCstation

4.5 連想辞書の日本語化

日本語テキストの場合，そもそも単語を切り出してくることが難しい．電子ニュース記事には，通常の辞書には載っていないいわゆる未知語も多く，またインフォーマルで会話調の文章が多いため，さらに問題が難しくなっている．そのため，第一版までは，英語のテキストからしか連想辞書を構築することができなかった．

---

[14] 引用行の先頭に，引用記事の投稿者の名前を書く方法など．
[15] ヘッダのうちサブジェクト中の単語は普通にカウントする．
[16] ヘッダの処理に関しては，第0版から実現されていた [3]
[17] 変化形の対応 (品詞情報を用いない方法) とハイフンで結ばれた複合語の分割に関しては，第0版から実現されていた [3]
[18] 規則変化ルールと不規則変化パターンを使用して行なう．品詞情報を用いる方法と用いない方法の2通りある．
[19] 接尾語による派生語生成ルールを使用して行なう．派生語の統一の過程で品詞の推定が可能になる．

第二版では，以下に示すような比較的単純な日本語処理により，日本語テキストから単語の切り出しを行なえるようにした．

テキスト中の漢字・カタカナを文字列の開始点として，漢字・カタカナが続く限り読み続ける．平仮名が来た場合には，あらかじめ仮名漢字変換システム用の辞書[20]を利用して作成しておいた単語リストを参照し，リスト中に載っているところまで読み続ける．このようにして日本語文字列の頻度付きのリストを作成していく．全テキスト情報を読み込んだあとで，各文字列の頻度情報を参考にしながら，単語へと分割していく．活用形の統一化は，単語リストをもとに行なう．

単語リストに載っていない単語は未知語となるが，そもそも単語リストは平仮名を含んだ文字列の切り出しと活用形の統一化にしか利用していないため，漢字とカタカナのみからなる単語に関しては特に問題は起こらない．また，この方法では，会話調の文章だからといって特に精度が落ちることはない．

なお，この方法では，平仮名から始まる単語を切り出すことができない．しかし，実際に調べてみると，平仮名から始まる単語には，重要度の値が小さいもの，すなわち一般的に出現頻度が高いものが多く，連想検索への影響はあまりない．

4.6 連想辞書の規模

第二版で使用している連想辞書の規模は，大体以下に示す通りである．

情報ソース：	電子ニュース (英語)
使用テキストの量：	約 1.5GB(約一カ月分の記事)
使用テキストの数：	約 200,000
単語の総数：	約 150,000
連想辞書の大きさ：	約 300MB
検索時間：	一秒以内〜数分
情報ソース：	電子ニュース (日本語)
使用テキストの量：	約 100MB(約一カ月分の記事)
使用テキストの数：	約 15,000
単語の総数：	約 110,000
連想辞書の大きさ：	約 20MB
検索時間：	一秒以内〜十数秒

検索時間は，検索方法・検索入力によって大きく変わってくる．検索方法 2，検索方法 3 では，検索入力に関わらず，どちらの連想辞書を使った場合でも一秒以内で検索が終了する．検索方法 1 と検索方法 4 では，平均的に前者の方が速い．ただし，両方法とも検索入力によって検索時間が大きく変わる．転置マトリックスファイル[21]を利用すると検索時間はほぼ半分になる．また，5.4節で説明する AND 的検索や，5.6節で説明する分野によるフィルタリング機能を用いると，検索時間は短くなる．逆に，5.2節で説明する推移的連想検索を行なうと検索時間は大幅に伸びる．

5 連想検索

連想検索は，Keyword Associator のサブシステムである kaserver によって行なわれる．本章では，まず，連想検索の仕組みについて簡単に説明した後，第二版において実現・強化された諸機能を順に説明する．

5.1 連想検索の仕組み

入力テキスト情報に現れる全ての単語の集合の上に，各単語を単位ベクトル w とする n 次元 (n は単語の総数) のベクトル空間を考える．この空間の中で，テキスト t を，重要度 $s_t(w)$ を用いて以下のようにベクトルとして表現し，

$$t_t = (s_t(w_1), s_t(w_2), \ldots, s_t(w_n))$$

大きさが 1 になるように正規化しておく．

$$t = \frac{t_t}{|t_t|}$$

次に，関連度 (Relevance) をベクトルの内積を用いて以下のように定義する．

$$\begin{aligned} R_{wt}(w_i, t_j) &= w_i \cdot t_j \\ R_{tw}(t_i, w_j) &= t_i \cdot w_j = R_{wt}(w_j, t_i) \\ R_{tt}(t_i, t_j) &= t_i \cdot t_j \\ R_{ww}(w_i, w_j) &= \sum_{k=1}^{m} R_{wt}(w_i, t_k) R_{tw}(t_k, w_j) \\ &= \sum_{k=1}^{m} (w_i \cdot t_k)(t_k \cdot w_j) \end{aligned}$$

ここで，R_{wt}, R_{tw}, R_{tt}, R_{ww} は，それぞれ，単語とテキスト，テキストと単語，テキストとテキスト，単語と単語の間の関連度を，m はテキストの数を表す．

連想検索は，この関連度 R の値を計算することによって行なわれる．すなわち，検索方法 1 なら R_{ww}，検索方法 2 なら R_{wt}，検索方法 3 なら R_{tw}，検索方法 4 なら R_{tt} をそれぞれ計算し，関連度の値によってソートすることによって，検索の結果が得られる [4]．

5.2 推移的連想

GrIPS のデモンストレーションのために，百人一首を入力テキストとして連想辞書を構築する機会があった[22]．

百人一首の歌の中には「冬」という単語は一度しか現れない．したがって，検索方式 2 によって得られる関連テキストは，「冬」が現れる 28 番歌のみとなり，また，検索方法 1 で得られる関連キーワードは，この歌の中に現れる単語のみとなる．この 28 番歌を入力として，検索方法 4 を行なってみると，31 番歌が，最も関連のあるテキストとして提示される[23]．この歌が選ばれたのは，28 番歌と 31 番歌の両方に「里」という単語が含まれていたことによる．続いて，31 番歌を入力として，検索方法 3 を行なってみると，関連キーワードとして「雪」が提示される．以上のような操作により，「冬」→「里」→「雪」という連想関係を導き出すことができたわけである．

推移的連想とは，上記のように連想を次々とたどること[24]によって，間接的な連想関係を導き出し，連想の多様性を増すための方法である．以下に，推移的連想の求め方を示す．

まず，テキストベクトル t を用いて，以下に示す連想マトリックス A_{wt} を作る．

$$A_{wt} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_3 \end{pmatrix}$$

[20] Wnn の辞書を使用した．

[21] 転置マトリックスファイルを合わせると，連想辞書の大きさはほぼ倍になる．

[22] テキストの全体量が少ないこともあり，一つ一つの歌を別々のテキストとして扱い統計処理を行なった．

[23] 関連度の計算結果では，自分自身，すなわちこの場合には 28 番歌がトップに来るが，結果を提示する際に，検索入力自身が含まれないようにしている．なお，この 31 番歌は，冬の歌として知られている．

[24] 推移的連想ではテキストの間の連想をたどっていく．すなわち，
検索方法 1：単語ーテキストー……ーテキストー単語
検索方法 2：単語ーテキストー……ーテキストーテキスト
検索方法 3：テキストーテキストー……ーテキストーテキスト
検索方法 4：テキストーテキストー……ーテキストー単語
というたどり方をする．

次に，この A_{wt} を用いて，連想を l 段階たどった際の間接的な連想関係を表す連想マトリックスを，以下のように定義する．

$$
\begin{aligned}
A_{wt} &= (A_{wt}A_{wt}^T)^l A_{wt} \\
A_{tw} &= A_{wt}^T (A_{wt}A_{wt}^T)^l \\
A_{tt} &= (A_{wt}A_{wt}^T)^l A_{wt}A_{wt}^T \\
A_{ww} &= A_{wt}^T (A_{wt}A_{wt}^T)^l A_{wt}
\end{aligned}
$$

ここで，$A_{wt}, A_{tw}, A_{tt}, A_{ww}$ は，それぞれ，単語とテキスト，テキストと単語，テキストとテキスト，単語と単語の間の推移的連想関係を表している．

推移的連想検索は，これらの連想マトリックスに，入力単語あるいはテキストに対応する列ベクトルをかけることにより求められる．すなわち，検索方法1なら $A_{ww}w^T$，検索方法2なら $A_{wt}w^T$，検索方法3なら $A_{tw}t^T$，検索方法4なら $A_{tt}t^T$ をそれぞれ計算すればよい．なお，この計算は5.1節の計算の拡張になっており，推移レベル l を0にすれば両者は一致する．

5.3 複数入力に対する重みの自動設定

Keyword Associator では，複数のキーワードあるいはテキストを入力として検索を行なうことができる．その際各入力に対して，ユーザの関心の度合を示す重みを与えることができる．検索結果は，各入力に対する結果を，与えられた重みによって重み付けして加算したものとなる [4]．

第一版では，重みを与えるためのインタフェースとして，スライダーが用意されていたが，入力のたびに毎回重みを設定するのも面倒であり，実際には，全てを同じ重みにすることの方が多かった．そこで，第二版では，kaeditor 上のリンク情報を利用して自動的に重み付けを行なうリンク検索の機能を実現することにより，煩わしい重み付けの作業をしなくても済むようになっている．

リンク検索の機能を働かせると，kaeditor からドラッグ&ドロップして入力したキーワードの他に，そこからリンクでたどれるものも一緒に入力とする．その際，リンクをたどった回数をもとに各キーワードの重みを計算する．すなわち，ドラッグ&ドロップしたキーワードの重みを1とし，そのキーワードからリンクを n 回たどることによって到達できるキーワードの重みを減衰係数の n 乗として計算する．減衰係数の値は，katool の設定ウィンドウ内のスライダーを用いて自由に調節することができる．また，リンクをたどる回数の最大値 (リンク検索範囲) を指定することも可能である．

なお，複数入力はリンク機能を働かせなくても可能である．複数のキーワードあるいはテキストをドラッグ&ドロップする，または，空白で区切ってキーボードから入力すればよい．この場合，各入力に対する重みはすべて1になる．

5.4 複数入力に対する検索結果の結合

第一版では，複数入力に対する検索結果は，各入力に対する結果のベクトルを (重み付け) 加算したものとなる．すなわち，OR 的な検索が行なわれるわけである．ところが，場合によっては，AND 的な検索，NOT 的な検索を行ないたいこともある．

第二版では，各入力に対する結果のベクトルの各要素ごとの積をとったものを計算することにより AND 的検索を，また，各入力に対する結果のベクトルの差をとったものを計算することにより NOT 的検索を実現している．

入力の間が「+」で結ばれていれば OR 的検索を，入力の間が「*」で結ばれていれば AND 的検索を，入力の間が「−」で結ばれていれば NOT 的検索を行なうようになっており，これらを組み合わせて使うことも可能である．

なお，入力の間が空白である場合，あるいはドラッグ&ドロップにより入力した場合には，デフォルトの検索方式が使用される．デフォルトは OR 的検索あるいは AND 的検索のいずれかを選択することが可能であり，katool のコントロールパネルの中で設定することができる．また，kaeditor, katool 上で，ドラッグすべきキーワードあるいはテキストを，シフトキーを押しながらセレクトすることにより，直前にセレクトしたものとの間で NOT 的検索を行なうよう指示することもできる．このように，キーボードから検索式を入力しなくても，単純な組合せであれば，簡単なマウス操作によって指定することが可能になっている．

5.5 ユーザへの適応

3.3節で述べたように，連想検索の結果として，ユーザにとって当たり前のものが得られると，それだけ役に立つものが減り，アイデアを広げていく刺激を受けるチャンスが減ってしまう．当たり前のものはできるだけ提示されないようにしなければならない．

ユーザにとって当たり前の連想関係にも2種類ある．

一つは，すでに Keyword Associator が提示した連想関係である．例えば，a というキーワードを入力したら b というキーワードが提示されたとしよう．b が，アイデアを広げていくきっかけになるものであったとしても，それは，はじめてみた時にそう思うのであって，2回目以降はあまり効果がない．つまり，再度 a に対する連想検索を行なった場合，また b が提示されるのでは，アイデアを広げていくという目的には不十分なのである．また，b を入力として a が提示されるのも，ユーザにとっては十分に予想されることであり，これもまた刺激としての意味はあまりない．

第二版では，ユーザの検索履歴を利用して，一度提示された連想関係が，何度も繰り返して提示されることがないようにするための，以下のような仕組みが実現されている．まず，検索が行なわれるたびに，検索入力と検索結果をセットにした検索履歴を記録していく．検索結果を出力する際に，検索入力と検索結果のペアが過去の検索履歴の中に存在するかどうかを調べ，存在する場合にはその回数に応じて関連度の値を小さくする．こうすることにより，システムを使っていくうちに次第にユーザにとって当たり前なものが排除されるようになり，絶えず新鮮な刺激を受けることができるようになる[25]．

なお，検索履歴には検索が行なわれた時刻も記録されている．したがって，ある一定の時間が経過した履歴に関しては無視する，すなわち，前回提示してからある一定の時間が経ったものに関しては再度提示させるようにすることもできる．また，各履歴から現在までに経過した時間に応じて関連度の下げ幅を変える，すなわち，最近提示されたものは関連度を大きく下げ，時間が経ったものはあまり下げないようにすることもできる．こうすることによって，忘れた頃にまた提示するといったことも可能になる．

Keyword Associator を使うことによって新たに得られた連想関係については，上記の方法で何とか対処することができる．が，そもそも，ユーザは Keyword Associator を使う前から，さまざまな連想関係を当たり前だと思っており，それもまた排除してやる必要がある．kabuilder はテキスト情報から連想関係を抽出し，連想辞書を作成する．これと同じことをユーザが作成した文章に対して行なう．すると，そのユーザがすでに持っている連想関係を抽出することが可能になり，その情報を用いて，検索結果の中から当たり前のものを減らすことができるようになる．むろん，文章の中に表されたものしか利用することができないので，

[25] 第0版を開発していた段階から，このような目的の機能の必要性が指摘されており，今後の課題として挙げられていた [3]．そして，第一版において，現在のものと似た機能が実現された [4]．ただし，検索結果のみを用いた方法であったため，第二版のものに比べると効果が弱かった．

十分とはいえないが，最近では，文章を書く作業もエディタやワープロを使って計算機上で行なうケースが増えてきているので，ある程度の効果をあげることができる．

5.6 分野によるフィルタリング

電子ニュースから構築した連想辞書には，さまざまな分野における連想関係が含まれている．これらすべての分野を利用する場合には，各分野における連想関係がマージされて提示されることになる．これを，各分野ごとに別々に眺めることによって，また新しいことが見えてくる場合もある．

第二版では，特定の分野を指定し，その中で見つかる連想関係だけを利用した連想検索を行なうことができるようになっている．5.1節で説明したように，Keyword Associatorでは，テキストベクトル t を利用して連想検索を行なう．したがって，このテキストベクトルのうち特定の分野に関するものだけを利用すれば，その分野に現れる連想関係だけを検索することができるようになる．

分野の指定は，検索に利用するテキストベクトルの名前を正規表現の形で指定することによって行なう．kabuilderは，連想辞書を構築する際に，各テキストベクトルに名前をつけている．電子ニュース記事を利用した場合，各記事に対するテキストベクトルの名前は，ニュースグループ名，記事番号，投稿された日時をもとに生成される．したがって，例えば，「soc.*」という指定を行なえば，「soc」という社会問題を扱うニュースグループの中にある記事に対応するテキストベクトルのみが検索に使用されるようになる．このようにして，特定のニュースグループ，すなわち特定の話題において特徴的に現れる連想関係を扱うことができる．

また，検索方法2あるいは検索方法4により得られる関連テキストを利用して分野を指定することもできる．katoolにある「ホールド」ボタンを押すと，それ以降の連想検索を，現在得られている関連テキストに対応するテキストベクトルのみを利用して行なうようになる．関連テキストは，検索入力に関連する分野のものが選ばれてきている．したがって，関連テキストを利用することにより特定の分野を指定することができる．

5.7 品詞によるフィルタリング

第二版では，連想検索の結果として特定の品詞だけを出力するようにするためのフィルターが用意されており，動詞だけ，形容詞だけを提示するといった使い方ができる．

発散的思考のための発想法の中に，属性列挙法という方法論がある．対象としている問題を，属性によって分割し，各属性についてアイデアを出していくことにより発想を広げる方法である．属性の分類としては，名詞的属性・形容詞的属性・動詞的属性に分けるやり方がよく知られている．例えば，車について考える時に，車の「動作」(動詞)に着目してアイデアを出したり，車の「性質」(形容詞)に着目してアイデアを出したりするわけである．品詞によるフィルタリングを利用することにより，このような発想の方法を支援することも可能である．

6 おわりに

本稿では，第二版で実現・強化された諸機能を中心に，システム構成，ユーザインタフェース，連想辞書構築，連想検索という4つの側面から Keyword Associator の紹介を行なった．第二版での改良点を，第1章の最後に列挙したが，これらをまとめると，以下の5点に集約することができる．

パフォーマンスの向上
: サーバ・クライアント・アーキテクチャを採用し，連想検索を高速化することにより，システム全体のパフォーマンスを向上させ，大規模な連想辞書にも対応できるようにした．

操作性の向上
: ドラッグ&ドロップを中心とした DMI を全面的に採用し，ほとんどの機能を簡単なマウス操作により実行できるようにした．また，通信機能により，他のツールと組み合わせて使用することもできるようにした．

連想辞書の強化
: テキスト単位での統計処理により，多様な連想関係を持つ大規模な連想辞書を構築できるようにした．また，日本語のテキスト情報からも連想辞書を構築できるようにした．

ユーザへの適応
: ユーザに適応し，ユーザにとって当たり前な連想関係が，できるだけ提示されないようにするための機能を強化した．

連想検索の多様化・強化
: 連想関係を次々とたどることにより連想の多様性を増すための推移的連想検索，AND 的な検索や NOT 的な検索，分野によるフィルタリング機能，品詞によるフィルタリング機能を実現することにより，ユーザの意図・目的に応じて多様な検索方法を利用することができるようにした．

現在，以前から課題として挙げていた [3][4]，

- リスト形式以外の方法による検索結果の提示
- キーワードの使われ方の時間的変化の利用

という2点に関して検討・予備実験を行なっている．

また Keyword Associator の機能を，

- 情報の分類
- 情報の検索
- 情報のブラウジング
- 情報のフィルタリング [6]
- 収束的思考支援
- 要求獲得支援

に応用するための検討・予備実験も並行して進めている．

今後はこれらに加え，Keyword Associator の評価も行なっていく予定である．

参考文献

[1] 渡部 勇：アイデアプロセッサに関する基礎的考察，情報処理学会第 38 回全国大会予稿 2J-2，pp. 623-624 (1989).

[2] Watanabe, I. : IdeaEditor: Overview, Research Report IIAS-RR-90-2E, IIAS-SIS, FUJITSU LABORATORIES LTD. (1990).

[3] 渡部 勇：発散的思考の計算機支援，情報処理学会研究会報告 (ヒューマンインタフェース), Vol.90, No.18 (1990).

[4] 渡部 勇：発散的思考支援システム: Keyword Associator，計測自動制御学会合同シンポジウム論文集（1991 年 10 月 29 日～30 日・岡山），pp. 411-418 (1991).

[5] 神田陽治，渡部 勇，三末和男，平岩真一，増井誠生：グループ発想支援システム GrIPS，人工知能学会誌，Vol.8, No.5, pp. 65-74 (1993).

[6] 渡部 勇：緩い協調：協調情報フィルタリングシステム，情報処理学会研究会報告 (ヒューマンインタフェース), Vol.91, No.18, pp. 179-186 (1991).